Connecting Wireless Sensor Networks to the Internet
MSc. Thesis Proposal

Matúš Harvan
m.harvan@iu-bremen.de

Fall Semester 2006

Supervisor: Jürgen Schönwälder

International University Bremen [1]
Campus Ring 1
28759 Bremen
Germany

[1] *The Unseen University* as of Spring 2007

Abstract

The focus of the proposed work is to investigate mesh networking over 802.15.4 links in wireless sensor networks with the MicaZ and TelosB motes running the TinyOS operating system. Background information on several related areas is presented, such as the TinyOS operating system and motes, the RoboCube platform and CubeOS operating system, the 802.15.4 and 802.15.5 standards, the uIP TCP/IP stack implementation, the 6lowpan approach for transmitting IPv6 traffic over 802.15.4 links and the Delay-Tolerant Networking.

Contents

Chapter 1

Introduction

Wireless sensor networks consist of numerous tiny nodes equipped with various sensors and a radio interface for communication. Among the applications are environment monitoring such as forest fire detection and water or air quality monitoring, wildlife monitoring, smart spaces, medical systems and robotic exploration. Due to the nature of the application, access to the motes may not be feasible after initial deployment. Hence, the devices have to run for extended periods of time on battery power, resulting in low-power, energy-saving designs.

The purpose of the proposed work will be to investigate ways of connecting such wireless sensor networks to the Internet, i.e. enabling TCP/IP support, and focus on wireless mesh networking mechanisms over 802.15.4 links in wireless sensor networks composed of TinyOS-enabled motes.

As preparation for this work, several related areas have been studied in more depth and the collected background information fills the majority of this document. Chapter 2 gives an overview of several hardware platforms and operating systems suitable for wireless sensor network scenarios. Chapter 3 describes the underlying link layers (PHY and MAC) facilitating the communication in a wireless sensor network, focusing on 802.15.4 and the 802.15.5 mesh extensions. Chapter 4 describes several approaches to support the higher level network layers and protocols, such as IP, TCP, UDP and ICMP, in wireless sensor networks. Various modifications to make TCP/IP more viable for wireless sensor networks are also covered. The Delay Tolerant Networking approach, covered in Chapter 5 may also be applicable to wireless sensor networks. Chapter 6 lays out details of the proposed work to be done within the scope of a Master Thesis.

Chapter 2

Hardware Platforms and Operating Systems

Several embedded hardware platforms and corresponding operating systems suitable for wireless sensor network scenarios will be described. They can roughly be divided into tiny motes running the TinyOS system, RoboCube-based platforms running the CubeOS system and embedded devices running a UNIX-like system, such as Linux.

2.1 TinyOS

The most typical hardware platform used for wireless sensor networks are tiny, low-power motes. These are embedded systems with an 8-bit micro-controller, RAM and flash memory sizes in the order of kilobytes and physical sizes in the order of a few cm^2. The motes are also equipped with an RF interface. Spending most of the time in sleep modes, they can run for several years on 2 AAA batteries. Probably the most widely known are the motes originally developed at UC Berkeley and produced by the Crossbow Technologies company. For example, CrossBow's Mica2 motes use an Atmel ATmega128L micro-controller, have 4 KB of RAM and 128 KB of flash memory [9] and an RF interface operating 433 and 868-915 MHz frequencies. The MicaZ motes have an 802.15.4 compliant interface. Clearly, these motes are suitable for low data rate applications requiring only minimum data processing. However, target costs of less than 10 cents per mote would enable networks with potentially thousands of devices.

These motes typically run the TinyOS operating system [15]. It originated at UC Berkeley and was designed for extremely restricted devices such as the motes. It is an event-driven operating system supporting concurrency with low overhead. There are no blocking operations. All long-latency operations are *split-phase*, i.e. commands requesting an operation return immediately and completion of the operation is signaled with an event. The system provides a set of reusable components, which are combined together using the so-called *wiring specification*. The OS has a very small footprint, with the core OS requiring only 400 bytes of code and data memory.

The TinyOS operating system is written in the nesC language [15]. nesC is

a dialect of the C language. It is "static" language with no dynamic memory allocation and no dynamic dispatch. This allows for whole program analysis at compile time, resulting in efficient optimizations. Furthermore, safety checks, such as data-race detection are also performed at compile time.

2.2 RoboCube and CubeOS

RoboCube [2] is an embedded hardware platform used mainly in robotics. It is based on the Motorola MC68332 processor and has 1 MB of RAM and 1 MB of flash-EPROM. The hardware design involves various boards, such as processor-, bus- and I/O-boards, which can be stacked on top of each other to form an embedded system. A board's size is 77x86 mm. The bus-board provides SPI and I^2C controllers, allowing to attach various sensors. In a typical robotics scenario, energy for electric motors of the robot based on a RoboCube system is also supplied from the battery, resulting in lower battery life time. For example, battery life time of the RoboCube systems used in the Robotics Lab at International University Bremen is in the order of hours rather than years.

CubeOS [2] is an operating system for the RoboCube platform. It is a small, embedded, real-time operating system providing preemptive and cooperative multi-threading; thread create, suspend, resume, sleep and kill functions; IPC with signals, semaphores and message queues; data primitives such as lists and buffers; control primitives such as reactive and PID control; sensor and actuator abstraction; layered radio communication protocol; real-time clock; and various device drivers. CubeOS consists of about 30 KB of binary code and the modular design allows to compile together only the necessary components for a specific RoboCube hardware configuration.

The robotics framework is further aided by the RobLib library building on top of the CubeOS system. It provides medium to high level functions for motor control and supports dead-reckoning and motion control.

Although the RoboCube platform with CubeOS system is used mainly in robotics, it could be equipped with various sensors and used also in wireless sensor network scenarios.

2.3 Unix-like systems

Several UNIX-like operating systems such as Linux or NetBSD are available also for various embedded platforms. Using these operating systems may be an appealing alternative to a custom operating system given the plethora of readily available applications.

For example the SeNDT project [29] at Trinity College Dublin uses an Intel XScale-based platform with 64 MB RAM and 64 MB flash memory running ARM Linux in a wireless sensor network application for monitoring lake water quality. The battery life of these devices reaches half a year.

It should be noted that a port of the Linux operating system, named uCLinux, exists also for several architectures without a memory mapping unit. Hence, the Motorola MC68332-based RoboCube platform could also use the Linux operating system rather than CubeOS.

Chapter 3

PHY and MAC Layers

Wireless sensor network devices usually have a built-in radio interface and communicate using the wireless channel. There are several possibilities for the underlying wireless communication technology.

The early motes were not using any standardized PHY and MAC layer protocols. The radio interface used two-tone Frequency-Shift-Keyed (FSK) modulation at 433 and 868-915 MHz supporting data rates up to 38.4 kbps. In the case of TinyOS, the operating system provides an addressing scheme and takes care of physically transmitting messages over the medium without any standardization of the PHY or MAC layer protocols. Interoperability basically requires using the same operating system and cross-vendor hardware compatibility is unclear.

The IEEE 802.11 family provides several standardized protocols (802.11a, 802.11b, 802.11g) for wireless communication and compatible devices are widely used. However, these protocols provide unnecessarily high range and data rates for wireless sensor network scenarios, resulting in high energy demands for wireless communication. Therefore, they are not well suited for the tiny motes with limited resources and battery power.

3.1 802.15.4

A standardized alternative targeted at limited devices such as the tiny motes is the 802.15.4 standard [19]. It was developed by the 802.15.4 Task Group within the IEEE and defines the physical layer (PHY) and medium access control (MAC) layer specifications for low data rate wireless personal area networks (LR_WPANS). Such networks are typically limited to a personal operating space (POS) of up to 10 meters and involve little or no infrastructure. The standard provides for low complexity, low power consumption, low data rate wireless connectivity among a wide range of inexpensive devices. Among others, wireless sensor networks seem to be a suitable application scenario for 802.15.4 networks.

3.1.1 Network topologies

An 802.15.4 network consists of two types of devices, full-function devices (FFD) and reduced-function devices (RFD). An FFD can operate as a personal area

network (PAN) coordinator, a coordinator or a device while an RFD can only act as a device. An FFD can talk to RFDs and other FFDs, while an RFD can only talk to FFDs. An RFD is intended for very simple applications, such as a light switch or a passive sensor, with no need to send large amounts of data. An RFD may associate with only one FFD at a time. As a result of these restrictions, an RFD needs only minimal resources and memory capacity.

An 802.15.4 network is constituted of at least two devices within a POS communicating on the same physical channel. It shall contain at least one FFD acting as the PAN coordinator. The network may operate in a star or a peer-to-peer topology.

In the star topology devices communicate with a single central PAN coordinator. While RFDs act as communication end-points only, a PAN coordinator mainly routes communication around the network. Different star networks operate independently of each other. This is achieved by using a PAN identifier which is unique within the radio range. After the PAN coordinator has chosen a PAN identifier, it can allow other devices, both FFDs and RFDs, to join the network.

The peer-to-peer topology also has a PAN coordinator, but additionally devices can communicate directly with each other. This allows for more complex topologies, such as a mesh topology or a cluster-tree.

The cluster-tree network is a special case of a peer-to-peer network with mostly FFD devices. As a RFD can associate to only one FFD, RFDs can participate in cluster-tree networks only as leave nodes. Any of the FFDs can act as coordinators and provide synchronization services to other devices and coordinators. One of these coordinators becomes the overall PAN coordinator. The PAN coordinator forms the first cluster by picking an unused PAN identifier, becoming the cluster head (CLH) with cluster identifier (CID) of zero and broadcasting beacon frames. Devices receiving these beacon frames may request joining the cluster at the CLH. If the PAN coordinator grants the joining, the new device will be added to the PAN coordinator's neighbor list and start broadcasting beacons as well. Other devices may then join the network at this new device as well. If it is not possible to join the network at the CLH, a device searches for another parent device. The PAN coordinator may instruct another device to become the CLH of a new adjacent cluster. This could happen if predetermined application or network requirements are fulfilled. As other devices connect, a multicluster network structure is formed. In its simplest form, the cluster tree network consist of only a single cluster, but larger networks may be formed as a mesh of multiple neighboring clusters. This is illustrated in Figure 3.1. The multicluster structure trades increased message latency for an increase in coverage area. Such a peer-to-peer network can clearly be ad-hoc, self-organizing and self-healing.

The 802.15.4 standard provisions for two types of addresses. All devices shall have unique 64-bit extended IEEE addresses. These extended address can be used for direct communication within the PAN. Additionally, devices can be allocated 16-bit short addresses by the PAN coordinator during association with the PAN coordinator.

3 frequency bands using different data rates are available for 802.15.4. They are summarized in Table 3.1.

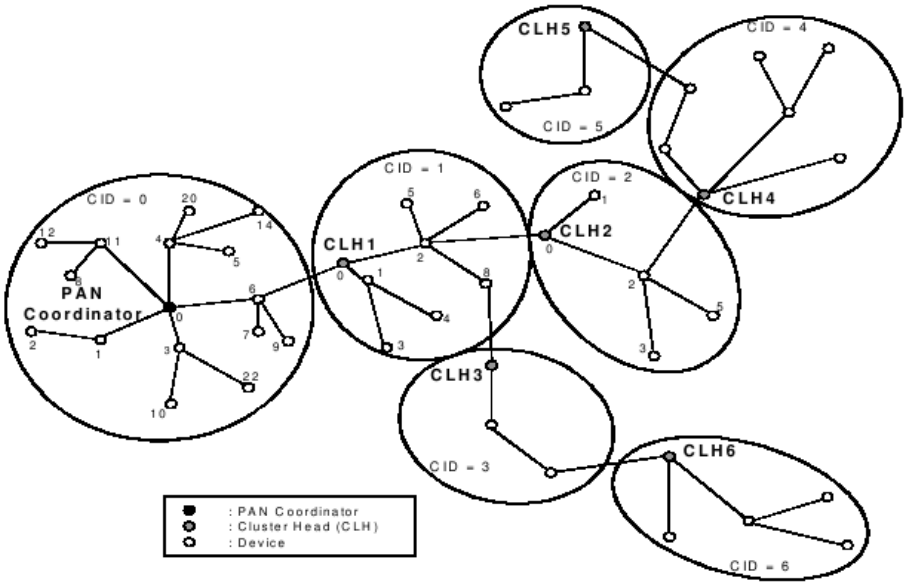

Figure 3.1: Cluster Tree Network – lines represent parent-child relationships rather than communication flow. Image taken from [19].

frequency	data rate
2400 – 2483.5 MHz	250 kbps
902 – 928 MHz	40 kbps
868 – 868.6 MHz	20 kbps

Table 3.1: 802.15.4 frequency bands

3.1.2 Data transfers in beacon-enabled and non-beacon networks

There are two modes of operation of an 802.15.4 network, a beacon-enabled and a non-beacon mode.

In the beacon-enabled mode, an optional superframe structure is used. It is defined and bounded by the beacons broadcasted by the coordinator. The beacons are used to synchronize devices, identify the PAN and describe the superframe structure. The superframe is divided into 16 equally sized slots. A beacon is broadcasted in the first slot of each superframe. An illustration is available in Figure 3.2. Devices wishing to communicate during the contention access period (CAP) between two beacons compete with other devices using s slotted CSMA-CA (Carrier Sense Multiple Access - Collision Avoidance) mechanism. The superframe may be divided into an active and an inactive portion. During the inactive portion the coordinator does not interact with the PAN and may enter a low-power mode. The coordinator may dedicate portions of the active superframe to guaranteed time slots (GTSs) for low-latency applications or bandwidth guarantees. The GTSs form the contention-free period (CFP) and appear at the end of a superframe. There may be at most 7 GTSs per superframe and a GTS may occupy more than one time slot.

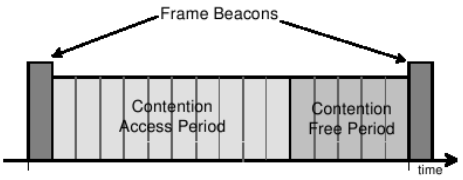

Figure 3.2: Superframe structure. The Contention Free Period is optional. Image taken from [19].

There is a difference between data transfers from a device to a coordinator and vice-versa. For data transfers from a device to a coordinator, the device uses slotted CSMA-CS to transmit its data frame. For data transfers from a coordinator to a device, the coordinator indicates in the beacon that data is pending for the device. The device periodically listens for the beacon broadcasts. If a data transfer is pending for it, it transmits a MAC command requesting the data transfer. This MAC command is transmitted using slotted CSMA-CA. Upon reception of the MAC command, the coordinator uses slotted CSMA-CA to transmit the data frame to the device.

The coordinator may decide that non-beacon mode is used. Then there is no superframe structure and devices use unslotted CSMA-CA instead of slotted CSMA-CA. Note that beacons are still needed for network association. For data transfer from a coordinator to a device, the device has to request the data transfer using a MAC command. If there is data pending for the device, it is transmitted in a data frame. Otherwise, a data frame with zero-length payload is transmitted, indicating no pending data for the device.

For peer-to-peer data transfers the devices either receive constantly or synchronize with each other. In the former case unslotted CSMA-CA is used while the latter case is beyond the scope of the 802.15.4 standard.

The 802.15.4 protocol has been designed to favor battery-powered devices. These can spend most of their time in a sleep state saving battery power. However, they have to periodically wake up and check if there are any messages pending by listening to beacons. This allows the application designer to balance between battery consumption and message latency.

3.1.3 Robustness

Robustness in the 802.15.4 networks is achieved by using optional frame acknowledgments, CSMA-CA mechanisms and data verification.

The 802.15.4 standard accommodates optional frame acknowledgments for MAC command frames and data frames. Note that in both non-beacon and beacon-enabled networks, these acknowledgments are sent directly without using CSMA-CA.

The non-beacon networks use an unslotted CSMA-CA channel access mechanism. When a device wishes to transmit a frame, it has to wait for a random period of time. If the channel is idle after this random period of time, data shall be transmitted. In case the channel is busy, the device shall wait for another random period of time before retrying.

Beacon-enabled networks use a slotted CSMA-CA channel access mechanism. The backoff slots are aligned with the start of the beacon transmission.

A device wishing to transmit during the CAP period waits for a random number of backoff slots. If the channel is idle afterwards, it can transmit. Otherwise, it waits for another random number of backoff slots before retrying.

For the data verification part, a 16-bit cyclic redundancy check (CRC) is used on every frame to detect bit errors.

3.1.4 Security

Several security services such as maintaining an access control list (ACL) and using symmetric-key cryptography to protect transmitted frames are specified by the standard.

Using these services, devices may operate in one of the three security modes: unsecured, ACL and secured mode. In the unsecured mode, no security services are used. Devices operating in the ACL mode maintain ACL lists of devices from which they are willing to receive frames. Devices operating in the secured mode use cryptography services in addition to ACLs. The cryptography services include data encryption for beacon, command and data payloads, usage of a message integrity code to provide frame integrity, i.e. to protect data from being modified by parties not sharing the encryption key, and sequential freshness using an ordered sequence of inputs to reject replayed frames. The freshness checking works by comparing the freshness value of a received frame with the last known freshness value. If it is newer, the check has passed and the last known freshness value is updated. The distribution of the symmetric encryption keys is not specified by the 802.15.4 standard.

3.1.5 Implications for higher layers

From the viewpoint of higher network layers, an important aspect of 802.15.4 is its limitation on the frame size. The PHY header uses a 7 bit field to specify payload length in bytes (0-127 bytes). Taking into account the PHY and MAC layer headers, this leaves a Maximum Data Length of 102 bytes for the higher layers. Further interesting implications IP traffic are mentioned in [30]:

1. Links are predominantly bimodal for short packet bursts.
2. Sporadic traffic observes intermediate links, which are due to SNR variations.
3. There are ETX asymmetries, which are larger over longer time intervals.
4. Acknowledgement failures are correlated.

The 802.15.4 protocol is defined in the 802.15.4-2003 standard. This document was approved in May 2003 and published in October 2003. With releasing of the standard, the work of the 802.15.4 task group has been completed, the group was hibernated and two new task groups, 802.15.4a and 802.15.4b, have been formed.

3.1.6 802.15.4a Task Group

The 802.15.4a Task Group is developing an amendment to the current 802.15.4-2003 standard for an alternate PHY to provide high precision ranging and location capability with 1 meter accuracy and better, high aggregate throughput,

ultra low power, higher data rates, longer range, lower power consumption and lower cost. The baseline specification has been selected in March 2005 to include two optional PHYs consisting of a UWB Impulse Radio operating in the unlicensed UWB spectrum and a Chirp Spread Spectrum operating in the unlicensed 2.4GHz spectrum. The UWB Impulse Radio should be able to deliver high precision ranging. The final standard is expected to be published by IEEE in March 2007.

3.1.7 802.15.4b Task Group

The 802.15.4b Task Group is refining the current 802.15.4-2003 specification to clear up ambiguities and resolve inconsistencies. Furthermore, the group is supposed to make specific extensions such as a faster sub-GHz physical interface, add support for time synchronization, reduce unnecessary complexity, increase flexibility in security key usage and consider newly available frequency allocations. The IEEE 802.15.4b standard has been approved in June 2006 and is waiting for publication.

3.1.8 ZigBee Alliance

While the IEEE 802.15.4 standard provides the lower network layers, the ZigBee alliance [32] is supposed to provide the upper layers ranging from the network layer to the application layer, including application profiles. The alliance provides interoperability compliance testing and marketing of the standard. It intends to ensure cross-vendor compatibility, i.e. it should guarantee that a light switch from one company works with the lights from another company. The ZigBee standard has been publicly released in June 2005. In December 2005 there have been 6 compliant platforms.

3.2 802.15.5

Mesh networking is a mechanism providing multi-hop routing within the link layer. Within a wireless network, where not all nodes necessarily have connectivity to the next-hop router, IP routing would not be suitable. However, using multi-hop paths over intermediate nodes, the range of wireless networks can be further increased without modifications to the IP layer. A plethora of mesh networking protocols has been developed for mobile ad-hoc networks (MANETs) and a description of the more than 70 different protocols is beyond then scope of this document. However, the IEEE is developing the 802.15.5 standard for mesh networking in 802.15 WPANs, which will be described in more detail.

The IEEE 802.15.5 working group is chartered with providing Mesh Networking support for 802.15 WPANs. By combining Samsung and Philip's proposal, a draft for the baseline document has been created. While not all details have been resolved yet, the document provides some insights how Mesh Networking will be enabled for both the high data rate 802.15.3 and the low data rate 802.15.4 WPANs. For wireless sensor networks and the proposed work, the extensions for 802.15.4 may be relevant and hence will be described more closely.

Mesh networking for 802.15.4 is enabled by using the adaptive robust tree (ART) and the meshed ART (MART) mechanisms. In an ART, nodes are organized into a tree form. Each branch of the tree is assigned a block of consecutive addresses. Nodes keep information about nodes in their branches and use this information for routing decisions. The ART has three phases, initialization, normal and recovery phase.

During the initialization phase, nodes join the network and a tree is formed. This phase is functionally divided into two stages, association and address assignment. During the association stage nodes gradually join the network and a tree is formed. To allow nodes to limit the number of children it is willing to accept and to balance the number of children among nodes, an acceptance degree (AD) is used in response to an association requests. The AD has one of the four values: 3 – accept without reservation, 2 – accept with reservation, 1 – accept with reluctance and 0 – reject. An associating node should choose the node with the highest AD response.

After a branch reaches its bottom (a suitable timer can be used for this purpose), the number of nodes in that branch is counted in a down-to-top way. Whenever a node joins the network, it starts a timer. If no other node joins before the time expires, the node becomes a leaf node and sends a children number report frame to its parent. After a non-leaf node has received the report from all its children, it reports to its parent by summing up all children requests and its own requests. The node and requested addresses counting is illustrated in Figure 3.3. Within the report frame nodes also indicate the number of requested address. The possibility of allocating more addresses than nodes allows for a small number of nodes joining the network later.

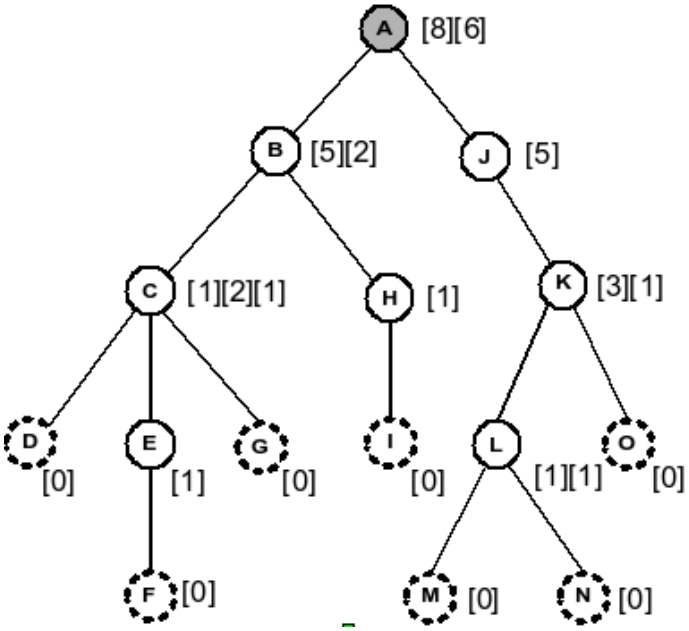

Figure 3.3: Calculating the number of nodes along each branch in an ART structure. The numbers indicate number of children within each branch. Image taken from [21].

After the root node receives information from all its branches, the address assignment stage starts. The root node assigns consecutive blocks of addresses to the branches and the addresses are distributed to the nodes in a top-to-bottom manner. After the address assignment is finished, a logical tree is formed and each node has a block addressing table (BAT) for tracking the branches below it.

After the initialization phase, the normal phase is entered. In this phase normal communication can start. The BAT table is used for routing decisions, i.e. to determine the next hop from a packet's destination address. If the destination address is not in one of the branches, the frame is forwarded to the parent. Small numbers of nodes can still join the network during the normal phase. For larger changes in the number of nodes or the tree topology, the tree has to be initialized again. Link and node failures can trigger the recovery phase for affected parts of the tree, with the rest of the tree continuing in the normal phase.

The ART structure can be extended to a Meshed ART by allowing connections between neighbors as illustrated in Figure 3.4. For this, the nodes broadcast hello messages to learn their neighbors and construct a connectivity matrix. Compared to an ART, the MART allows shorter routes and eliminates some single points of failure.

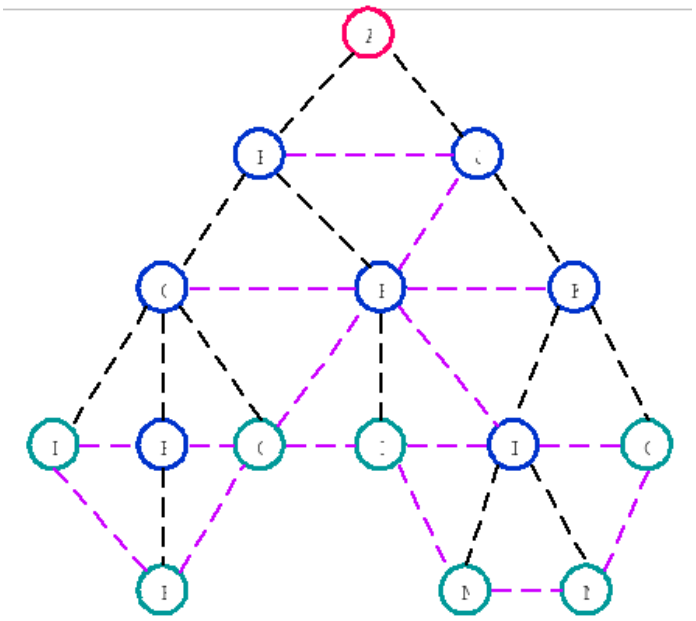

Figure 3.4: Meshed ART - the black lines represent links in an ART while the magenta lines represent the additional connections in a MART. Image taken from [20].

The [21] document details out the mechanisms, frame formats as well as an extension for multicast mesh routing.

Chapter 4

Above the Link-Layer

Many wireless sensor networks cannot be operated in isolation. They need to be accessible from external networks for monitoring and controlling. The ubiquity of TCP/IP has made it the de-facto standard protocol suite for computer networks. TCP/IP support in sensor networks would allow benefiting from the ease of interoperability and generality of an estabilished technology, a plethora of readily available applications and make it possible to connect wireless sensor networks directly to other networks and the global Internet. By TCP/IP support, UDP and ICMP protocol support is meant as well. While the UDP protocol is suitable for transferring sensor data, the TCP protocol is useful for configuration, management, re-programming and system updates of the sensor nodes. Several ways to enable IP connectivity and support for the TCP, UDP and ICMP protocols support in wireless sensor networks are possible. A proxy-based approach scheme and two approaches for native TCP/IP support on the sensor nodes, uIP and 6lowpan, will be described.

4.1 Proxy

In the simplest scenario a proxy-based scheme is used. A special proxy server is employed as a gateway between the sensor network and the TCP/IP network. As all communication between clients from the TCP/IP network and sensor network goes via the proxy, the two networks are separated and the communication protocol within the sensor network can be freely chosen.

The proxy can operate as a front-end or as a relay. In the former case, the proxy acts as a front-end for the sensor network, pro-actively collects data from the sensor nodes and stores the data in a database. Clients from other networks then query the proxy for sensor data. In the latter case, the proxy relays data between sensor nodes and TCP/IP hosts, possibly translating TCP/IP packets to a custom protocol used within the sensor network. One such approach, the Sensor Internet Protocol will be described in more detail.

It should be noted that a proxy can also be useful when sensor nodes natively support TCP/IP. Here, the proxy can offload the sensor nodes from resource-intensive tasks such as fragment reassembly. One such approach is described in [13].

4.1.1 Sensor Internet Protocol

The Sensor Internet Protocol (SIP) [24] is a proxy-based scheme for connecting wireless sensor networks with TCP/IP networks. The sensor nodes do not have any IP support. Instead, TinyOS Active Messages [16] are used within the wireless sensor network. An intermediate agent, a proxy, is acting as a gateway between external TCP/IP hosts and the Active Messages-based wireless sensor network. This proxy is assumed to be a more powerful device, which is not subject to severe computational and memory constraints such as the sensor nodes. The main motivation is to shift the burden of TCP/IP processing away from the motes onto the proxy.

As the motes do not have any notion of the TCP/IP protocol suite, the proxy translates between TCP/IP packets and Active Messages. In order to map IP addresses to actual node IDs, the proxy maintains a table with the corresponding mappings between IP addresses and node IDs. As IP fragment reassembly may consume a lot of resources, it is performed on the proxy. The Active messages used in TinyOS are limited to several tens of bytes, so the TCP/IP payload maybe be too large to fit into a single Active Message. This is left as an open problem in [24]. In general, IP options are dropped. The proxy handles also the ICMP, UDP and TCP protocols.

From the ICMP protocol, only echo, i.e. reply to ping is implemented. The reply is generated at the proxy rather being routed via the sensor network. Features such as Path MTU discovery or redirect are not supported.

As UDP is a connection-less datagram-oriented protocol, the translation is rather simple. However, for TCP the state of each connection has to be kept in the proxy. Furthermore, several optimization are done for the TCP protocol. The proxy acknowledges data on behalf of the motes. As the acknowledgement is sent prior to delivering data to the actual sensor, undelivered data may be acknowledged. While reducing round-trip times, this may be a problem with TCP/IP standards compliance as data that would never be delivered to the end host becomes acknowledged. The proxy also buffers and reorders TCP segments to the correct sequence. Restricted buffer space on the motes is taken into account and data may be queued on the proxy. Should transmission of data to the sensors be deferred, appropriate TCP congestion mechanisms and Explicit Congestion Notification are used. Finally, the burden of maintaining the TCP connection state is taken over by the proxy. It handles the opening and closing of TCP connections as well as TCP retransmissions. As with IP options, TCP options are also removed. It is assumed that the sensor nodes do not actively open connections and only accept incoming connections.

While not implementing the full TCP/IP functionality and ignoring TCP and IP options, the described scheme allows for basic TCP/IP communication between wireless sensors and external TCP/IP hosts. The communication is possible without modifying the wireless sensors or the external TCP/IP hosts. By moving most of the resources demanding tasks away from the tiny motes to the proxy, little resources are wasted on the sensor nodes.

4.1.2 Serial Forwarder

It should be noted that an even simpler proxy approach exists, but requires external hosts to be aware of the special communication protocol used within

the sensor network. All TinyOS Active Messages from external hosts are encapsulated into TCP/IP packets and forwarded by a gateway into the sensor network. An implementation is included with TinyOS under the name *serial forwarder*. The serial forwarder listens on a single IP address and TCP port. Active Messages are encapsulated as the TCP payload and forwarded by the proxy between the sensor network and the external IP network. As in the case of the previously described SIP, the sensors are not TCP/IP-aware. However, in contrast to SIP, this approach requires that applications on external hosts are aware of the Active Message protocol details and node IDs. An extension of the serial forwarder implementation to support IPv6 is described in [28].

4.2 uIP

uIP[11] is a TCP/IP stack implementation by Adam Dunkels. It runs on 8-bit controllers with a few hundred bytes of RAM. Only the minimal set of features needed for a full TCP/IP stack is implemented.

uIP deals with IPv4 only. It can only handle a single network interface. IP fragments are reassembled only for one packet at a time and they are dropped if not all segments are received within a specified time limit. From the ICMP protocol only echo has been implemented. Features such as Path MTU discovery or ICMP redirect are not supported. Opposed to the BSD socket API, the TCP API is event driven. This fits well into the TinyOS API design. There is no sliding window support as only one TCP segment per connection can be unacknowdledged, i.e. in flight. However, it should be noted that sliding window support is not required by the TCP specification. Another drawback of allowing only one unacknowledged packet is the negative impact of delayed acknodledgements[7]. A TCP receiver using delayed acknowledgements sends an acknowledgement only for every other received segment or in a time frame of at most 500ms if only one segment has been received. As uIP only sends one segment at a time, the receiver waits for as long as 500ms before acknowledging it. With only one TCP segment can be in-flight, congestion control mechanisms are not needed. The sent TCP segment is not buffered by the TCP stack and if retransmission is needed, the application is called. The UDP protocol is also supported.

The uIP implementation complies with all the TCP/IP requirements dealing with host-to-host communication as specified in RFC1122 [3], but falls short on the requirements for communication between the networking stack and the application. While additional care has to be taken when developing applications using uIP, no incompatibilities should arise when the uIP implementation communicates with other TCP/IP implementations.

4.2.1 TCP/IP Optimizations for Wireless Sensor Networks

In order to make TCP/IP more viable for wireless sensor networks several optimizations are suggested by the people involved with uIP in [12] and [4]. These optimizations will be described in more detail.

In large scale sensor networks, manual configuration of IP addresses would not be feasible and the DHCP mechanism is rather expensive in terms of required communication overhead. However, as sensor nodes can be assumed to know

their location, a spatial IP address assignment scheme would be possible. For example, the (x, y) location coordinates could be used as the two least significant octets of an IPv4 address. Furthermore, such a spatial addressing scheme would easily allow for regional broadcast mechanisms.

Energy savings can also be achieved by employing header compression mechanisms. As nodes in a sensor network usually share a common context, they could agree on common header fields. This would allow for efficient header compression mechanisms, reducing overhead and saving energy by having to transmit less data using the radio interface. A UDP/IP header compressor has been implemented reducing the UDP/IP header from 28 to 3 bytes [12].

The standard TCP/IP is known to have serious performance problems over wireless links [1]. The throughput problem has been addressed by e.g. the Snoop mechanism [1]. However, the low-power and energy-efficiency requirements of sensor networks add further constraints. The end-to-end retransmission mechanism employed by TCP is rather energy-inefficient in lossy wireless networks. In a multi-hop network, the retransmitted segment has to be forwarded by all intermediate nodes between the sender and the receiver, wasting energy at every hop. To alleviate this problem, the *Distributed TCP caching* (DTC) and *TCP Support for Sensor Nodes* (TSS) schemes have been proposed.

DTC

To avoid energy-costly end-to-end retransmissions TCP segments are cached at intermediate nodes between the sender and the recipient. A segment is retransmitted from an intermediate node having a copy of the lost segment in case of packet loss. Due to constrained resources of sensor nodes, each node caches only one segment. This is the segment with the highest sequence number seen so far with a certain probability. The probabilistic approach allows for older segments to be cached in the network as well. Only segments presumably not received by the next hop node are cached. Two mechanisms are available for detecting packet loss at the next hop. Either the link layer supports positive acknowledgements or the node overhears its successor transmitting the segment further. If a segment is presumed to be lost in transit, it is locked in the cache indicating that it should not be overwritten by another segment with a higher sequence number. A locked segment is removed from the cache upon receiving a TCP ACK acknowledging the cached segment or when the segment times out.

To avoid end-to-end retransmissions, DTC needs to respond to packet loss faster than the standard TCP. For this purpose DTC nodes maintain a soft TCP state for connections passing through them, measure the round-trip time (rtt) to the receiver and use a retransmission timeout of 1.5 rtt. The timeout value is then smaller for nodes closer to the receiver, allowing for the intermediate nodes to retransmit before the sender would do so. A retransmission timer is set when a segment is locked in the node's cache.

The TCP SACK option is used for both packet loss detection and as a signaling mechanism between DTC nodes. Upon reception of a TCP ACK with a smaller acknowledgement number than the node's cached segment's sequence number (`cached`), following actions are performed.

1. If `cached` is not in the SACK block, the cached segment is retransmitted and `cached` is added to the SACK block. If this action fills all gaps in the

SACK block, the acknowledgement can be dropped. However, a new ACK acknowledging all segments from the SACK block should not be generated as the receiver is allowed to discard a previously SACKed segment.

2. If `cached` is in the SACK block, the node can clear its cache as either the receiver has already received the cached segment or it is cached and locked by another node closer to the receiver.

Note that even if the sender or receiver does not support the SACK mechanism, the DTC nodes might add or remove the SACK option to enable SACK signaling for DTC.

While TCP data segments are cached, TCP ACKs are not cached as they can be easily regenerated. When an intermediate node encounters a TCP data segment, for which is has already forwarded a TCP ACK, it assumes the TCP ACK has been lost. Therefore, it does not forward the data segment and regenerates a corresponding TCP ACK.

To further aid DTC, the recipient may announce a small maximum segment size to avoid large TCP segments exceeding the sensor nodes storage capacity and announce a small window size to decrease the number of segments in flight.

TSS

Similarly to DTC, TSS tries to reduce the number of end-to-end retransmissions by caching TCP segments at intermediate nodes. However, it is completely based on TCP ACKs, so no link level acknowledgements are required. A segment is not forwarded until the successor has received all previous segments. This also prevents packet reordering, avoiding the need for resequencing buffers. Like with DTC, TCP segments are cached at intermediate nodes. A node always caches the TCP segment containing the first byte of data that has not yet been acknowledged or forwarded by the successor node towards the destination. A node knows that the successor has received a segment when it overhears the successor forwarding the segment further or the successor spoofs a TCP ACK acknowledging that segment. A segment known to be received by the successor will be removed from the cache. As with DTC, the cache holds only one packet. However, a buffer is required for temporarily storing packets waiting to be forwarded to the successor node. As with DTC, a retransmission is triggered by a timeout of 1.5 rtt.

TCP ACKs are not cached, but two mechanisms are employed to regenerate them. One is the same as used by DTC. The other one uses a timeout for acknowledgement regeneration if it does not overhear the acknowledgement to be forwarded by the successor node towards the TCP sender. The timeout is twice the average time measured between transmitting an acknowledgement to the successor and the successor forwarding it further.

For both DTC and TSS to work, no protocol changes are required at the sender or the receiver. However, symmetric and relatively stable routes are assumed for correct functionality. Furthermore, acknowledgements are modified, dropped and recreated in non-standard ways.

4.3 6lowpan

6lowpan is a working group within the IETF concerned with the specification of transmitting IPv6 packets over IEEE 802.15.4 networks. The IEEE 802.15.4 standard targets low-power wireless personal area networks (LoWPANs). Such LoWPANs consist of devices characterized by short range, low bit rate, low power and low cost. As a result, these devices typically are severely constrained and have only limited capabilities. 802.15.4 is covered in more detail in Section 3.1.

Currently, there are two 6lowpan internet drafts [23] and [25]. The former gives an overview, motivation and a problem statement while the latter dives into technical details and defines the frame format for transmission of IPv6 packets over 802.15.4 networks. Furthermore, creation of IPv6 link-local addresses and statelessly autoconfigured addresses on top of 802.15.4 networks is described. As IPv6 requires support of packet sizes larger than the maximum 802.15.4 frame size, an adaptation layer is defined. To make IPv6 practical on 802.15.4 networks, mechanisms for header compression and provisions for packet delivery in 802.15.4-based meshes are defined.

IEEE 802.15.4 defines four types of frames: beacon, MAC commands, data and acknowledgment frames. IPv6 packets are carried on data frames. It is recommended that these frames are acknowledged using the optional link-layer acknowledgment scheme of 802.15.4 to aid link-layer recovery. Use of the beacon-enabled 802.15.4 mode is not required for transporting IPv6 packets. Although not required by 802.15.4, for carrying IPv6 packets it is necessary to specify both source and destination addresses in the 802.15.4 frame header.

4.3.1 Addressing Modes

802.15.4 defines two types of addresses, IEEE 64-bit extended addresses and 16-bit short addresses unique within the PAN. Both types are supported by 6lowpan. However, 6lowpan imposes additional constraints on the short 16-bit addresses where specific prefixes have to be used depending on the type of the address. Unicast, multicast and reserved (for future use) prefixes are allocated in a new IANA registry.

Note that a 16-bit short address is only available after an association event. These short addresses addresses are rather transient in nature as their validity and uniqueness are limited by the lifetime of the association event and rely on the PAN coordinator. Hence, they should be used with caution.

It is assumed that a PAN maps to a specific IPv6 link, implying a unique prefix. Hence, the 16-bit PAN ID can be mapped to an IPv6 prefix. This can be used to implement IPv6 multicast by a link-layer broadcast limited to a PAN.

6lowpan also provides for stateless address autoconfiguration. As each 802.15.4 device has an EUI-64 identifier [18] assigned to it, an IPv6 interface identifier [17] can be obtained from this EUI-64 identifier using the stateless autoconfiguration described in [8].

Although all 802.15.4 devices have an EUI-64 address, it is also possible to use the short 16-bit addresses for autoconfiguration. In this case a pseudo 48-bit address is formed by concatenating the 16-bit PAN ID (or 16 zero-bits if unknown), 16 zero bits and the 16-bit short address from left to right. The result would then be

`16_bit_PAN_ID:16_zero_bits:16_bit_short_address`

The IPv6 interface identifier is formed from this 48-bit pseudo address using the stateless autoconfiguration [8]. However, the "Universal/Local" (`U/L`) bit should be set to zero in the resulting interface identifier to reflect that such identifier is not globally unique. Furthermore, all-zero addresses are not allowed in both cases.

The mapping of non-multicast (unicast) IPv6 addresses to 802.15.4 link-layer addresses follows the usual neighbor discovery in IPv6 as described in [26]. Packets with a multicast IPv6 destination address are sent to the 16-bit 802.15.4 address obtained by concatenating the 3-bit multicast prefix 101, bits 3 to 7 in the 15-th octet and the whole 16-th octet of the IPv6 address.

4.3.2 Adaptation Layer

The IPv6 protocol requires support for a Maximum Transmission Unit (MTU) of 1280 octets, which is well beyond the largest possible 802.15.4 frame size. Depending on overhead, the 802.15.4 protocol data units have different data sizes, leaving 81 to 102 octets for higher layers. Given the maximum physical layer packet size (`aMaxPHYPacketSize`) of 127 octets and a maximum frame overhead (`aMaxFrameOverhead`) of 25 octets, 102 octets are left at the MAC layer. Link-layer security imposes further overhead of 21, 13 or 9 octets in case AES-CCM-128, AES-CCM-64 or AES-CCM-32 is used, respectively. In the case of AES-CCM-128, only 81 octets are left available. This clearly is below the IPv6 MTU requirement, so an adaptation layer for fragmentation and reassembly is provided between layer two and three. This adaptation layer will be described in more detail.

IP datagrams transported over 802.15.4 are prefixed by an *encapsulation header*. It starts the 802.15.4 MAC protocol data unit (PDU) and is followed by the LoWPAN payload, e.g the IP header and payload. In case the `M` bit is set in the encapsulation header, a *Mesh Delivery* field follows before the LoWPAN payload. The encapsulation formats defined are referred to as the *LoWPAN encapsulation*.

The header starts with a 2-bit `LF` field, which indicates whether the frame is unfragmented (00) or it is the first (01), interior (11) or last (10) fragment. For the unfragmented case, a 2-octets long encapsulation header is used while for the fragmented case the header is 4-octets long. If the datagram does not fit within a single 802.15.4 frame, fragmentation is used. Upon receipt of link fragments, the recipient reconstructs the original unfragmented packet. The fragment-type encapsulation header includes a field indicating the complete datagram length, allowing the recipient to determine the size of the reassembly buffer. Details of the fragmentation and reassembly as well as the encapsulation header fields can be found in [25].

The adaptation layer provides also additional functionality beyond just fragmentation and reassembly. It includes an 8-bit `prot_type` field indicating what type of datagram follows after the header. This provisions for header compression usage in higher layers, which is discussed in Section 4.3.3. For IPv6, the `prot_type` value is 1, while 2 indicates header compression. Assignments of `prot_type` values are handled by a IANA registry. Furthermore, the header contains fields provisioning for mesh delivery – the `M` bit can be set in the en-

capsulation header, indicating that a *Mesh Delivery* field precedes the LoWPAN payload. Mesh delivery is discussed in more detail in Section 4.3.4.

Although the main reason for the adaptation layer is IPv6 compliance, it is expected that most 802.15.4 applications will not produce large packets. Using appropriate header compression, such packets could well fit into single frames. Nevertheless, the protocols themselves do not restrict bulk data transfers.

4.3.3 Header Compression

Even though 81 octets are left in a 802.15.4 frame for IPv6, the IPv6 header alone is 40 octets long, leaving 41 octets for upper layers. In case UDP is used, which has a header of 8 octets, only 33 octets can be used for application data. Note that the adaptation layer described in Section 4.3.2 further decreases the available space by 2 to 4 octets. These severe space restrictions make the use of header compression almost unavoidable.

Compared to published work and standardized approaches to header compression, IPv6 over 802.15.4 differs in several ways. Existing work assumes many flows between two devices while in 6lowpan only one flow is expected most of the time. Taking into account the limited packet sizes, integrating layer 2 and 3 compression seems viable. Furthermore, 802.15.4 devices would be mostly deployed in multi-hop networks. These differ from usual point-to-point link scenarios where the compressor and the decompressor are in direct, exclusive communication with each other. If preliminary context is required, which is often the case, building it should not rely exclusively on the in-line negotiation phase, so that already the fist packet sent could be compressed.

As the header compression changes packet format, its usage is indicated by setting the `prot_type` field in the encapsulation header to 2. Any new packet formats required by header compression could define new values for the `prot_type` field and reuse the basic packet formats.

If compressing the headers results in alignment not falling on an octet boundary, the remainder after the compressed headers is padded with zeros until the next octet boundary.

6lowpan currently defines header compression for IPv6 and UDP headers. Both will be described in more detail.

IPv6 header encoding

IPv6 header compression is possible even without a context-building phase as devices already share some state by virtue of having joined the same 6lowpan network. In particular, following values can usually be compressed: the *Version* is IPv6, both *IPv6 Source* and *Destination Addresses* are link local and the last 64 bits can be inferred from MAC layer source and destination addresses, the *Packet Length* can be inferred from the layer two `Frame Length` field in 802.15.4 PPDU or from the `datagram_size` field in the fragment header if it is present, *Traffic Class* and *Flow Label* are both zero and the *Next Header* is one of UDP, TCP or ICMP. Only the 8-bit *Hop Limit* field always has to be carried in full. Depending on how well a particular packet matches the described common case, several fields may have to be carried “in-line”. The compression scheme used for the IPv6 header is called *HC1*. The compressed header starts with an 8-bit *HC1 encoding field*, which is followed by non-compressed fields.

Bits $0-1$ are used for the IPv6 source address and bits $2-3$ for the destination address. For each of the addresses, one bit indicates whether a link-local prefix is assumed or the prefix is carried in-line. The other bit indicates whether the interface identifier is derivable from the link-layer address or is carried in-line. For mesh routing, the corresponding link-layer address is in the `Mesh Delivery` field. Bit 4 indicates whether Traffic Class and Flow Label are zero or their full 8- and 20-bit values are being sent. Bits $5-6$ indicate whether the Next Header is UDP, TCP, ICMP or its full 8-bit value is being carried in-line. The last bit indicates whether HC2 encoding follows the HC1 encoding or no more header compression bits follow after the HC1 encoding. Bits $5-6$ also indicate which particular type of the HC2 encoding follows, i.e. a UDP, TCP or ICMP encoding. The HC1 encoding field is followed by the Hop Limit field, which is always present. Other non-compressed fields, if any, follow after the Hop limit field. After these follows the actual next header such as UDP, TCP or ICMP, as specified by the `Next Header` field in the original IPv6 header. Using the HC1 encoding, the common IPv6 header, as described, can ideally be compressed from 40 octets to 2, where one octet is used for the HC1 encoding and one for the Hop Limit. Details of the header layout and the meaning of various bit values can be found in [25].

UDP header encoding

While the HC1 encoding specifies compression for the IPv6 header, allowing the `Next Header` field compression for ICMP, UDP and TCP, further compression of each of the corresponding protocol headers is possible using the *HC2* encoding. At the moment, only UDP header compression is specified. TCP and ICMP header compressions are to be determined later. The UDP HC2 header compression is called *HC_UDP*. This compression only applies if bits $5-6$ indicate that the protocol following the IPv6 header is UDP and bit 7 indicates the presence of HC2 encoding. Following fields can be compressed in the UDP header: *Source Port*, *Destination Port* and *Length*. The UDP checksum is always carried in full. While the Length field can be deduced from information available in other headers, the port fields have to be carried in-line either in full or partially compressed. The HC_UDP header starts with an 8-bit *HC_UDP encoding field*. Bit 0 indicates whether the UDP source port is compressed to 4 bits or completely carried in-line. If compressed, the actual 16-bit port number is calculated as $P +$ `short_port`. P is a number to be determined and coordinated with a IANA registry. `short_port` is a 4-bit value carried in-line. Bit 1 is used for the UDP destination port with meaning analogical to the source port. Bit 2 indicates whether the length field is calculated from the Payload Length in the IPv6 header minus the length of extension headers between IPv6 and UDP headers or carried in-line. Bits $3-7$ remain reserved for future use. The HC_UDP encoding field is followed by non-compressed or partially compressed fields carried in-line. These have to be in the same order as they would appear in a normal UDP header, i.e. source port, destination port, length and checksum. The HC_UDP scheme allows compressing the UDP header from 8 octets to 4. Details of the header layout and the meaning of various bit values can be found in [25].

4.3.4 Provisions for Meshes

Although 802.15.4 networks are expected to commonly use mesh routing, the 802.15.4 standard [19] does not define such capabilities. Therefore, 6lowpan specifies provisions required for packet delivery in 802.15.4 meshes. In a mesh scenario, devices do not require direct reachability to communicate with each other. Instead, intermediate devices are used as forwarders towards the final destination. From the two devices, the sender is known as the *Originator* and the receiver as the *Final Destination*. In order to achieve mesh delivery capabilities, the link-layer addresses of the Originator and the Final Destination have to be included in addition to the hop-by-hop source and destination.

For this purpose, the encapsulation header includes the `M` and `B` bits. Setting the `M`-bit indicates that a Mesh Delivery field follows the encapsulation header. The *Mesh Delivery field* carries the Originator and Final Destination addresses, providing for both IEEE extended 64-bit addresses and short 16-bit addresses. The `B`-bit indicates whether the destination is a unicast or a multicast address. For the former, a *Unicast Mesh Delivery Field* is used, while the latter uses a *Broadcast Mesh Delivery Field* carrying additional information. Additionally, the Mesh Delivery field contains a `Hops Left` field, which is decremented at each node when the frame is forwarded. When a value of zero is reached, the frame is discarded. More details about the header layout and interpretation of various bit values can be found in [25].

For just using mesh forwarding, a device does not necessarily have to participate in mesh routing protocols. While the FFDs are expected to participate as mesh routers, RFDs can limit themselves to discovering FFDs and using them for all their forwarding in a manner similar to IP hosts using a default gateway for all off-link traffic. A full specification of mesh routing such as specific protocols, interaction with neighbor discovery or controlled flooding are out of the 6lowpan scope.

Chapter 5

DTN

The delay- and disruption-tolerant interoperable networking (DTN) embraces an occasionally connected network that may suffer from frequent partitions and may be composed of more than one divergent set of protocol families. The basis of the architecture comes from the Interplanetary Internet [22], which focused on deep space communication in high-delay environments. Such architecture can among others readily be extended to occasionally connected networks such as sensor-based networks using (scheduled) intermittent connectivity and terrestrial wireless networks where end-to-end connectivity cannot be maintained. The DTN architecture is formally specified by Internet drafts from the DTNRG research group within the IRTF. A higher-level overview can be found in [31], [14] and [5].

The DTN architecture defines an end-to-end message-oriented *bundle layer*. This layer exists above the transport layers of the underlying networks and below the application layers. Devices implementing this bundle protocol are called DTN nodes. The bundle layer employs persistent storage to deal with network interruptions. It involves a reliable hop-by-hop transfer of reliable delivery responsibility and an optional end-to-end acknowledgment scheme.

Applications send so called Application Data Units (ADU), where each ADU is transformed into one or more bundles. The relative order of the ADUs might not be preserved during transfer. The bundles wait in a queue until a communication opportunity is available. Hence, a sufficient amount of storage has to be available. This storage has to be persistent and robust in order to survive application and operating system restarts and crashes.

5.0.5 Endpoint Identifiers (EID)

In order to identify the communication endpoints, variable-length endpoint identifiers (EDI) are used. Each node has at least one unique EID. These use the general syntax of URIs. An EID is composed of a scheme and a scheme-specific part (SSP). The interpretation of an SSP is defined by the respective scheme. In contrast to DNS name to IP address early binding, EIDs use late binding. Hence, the binding does not necessarily happen at the source and it might be the case that the mapping for an EID is not known at the time the transmission is started. An application wishing to receive traffic for a specific EID has to register for that EID. Such a registration is persistent in the sense that it sur-

Custody Transfer Requested
Source Node Custody Acceptance Required
Report When Bundle Received
Report When Bundle Custody Accepted
Report When Bundle Forwarded
Report When Bundle Delivered
Report When Bundle Deleted
Report When Bundle Acknowledged By Application
Confidentiality Required
Authentication Required
Error Detection Required

Table 5.1: DTN Delivery Options

vives reboots. Furthermore, a registration may fail. For example, an attempt to register for an invalid EID would fail. An EID refers to a set of DTN nodes and a node can determine from an EID the minimum reception group (MRG) of an EID. The MRG is a minimum set of nodes, to which a bundle must be delivered in order to complete the data transfer. This allows to use EIDs for single nodes as well as for multicast and anycast groups. Due to the possible delays in receiving a registration for a multicast group EID, some nodes may have to act as archivers of multicast messages in case someone joins the multicast group later.

5.0.6 Priority Classes

Several priority classes are defined. In increasing importance they are bulk, normal and expedited. First, bundles of higher priorities are transmitted. However, the prioritization affects only bundles from the same source. Optionally, nodes may enforce prioritization even across different sources.

5.0.7 Delivery Options

Applications can set various delivery options for ADUs. The delivery options can be used to track the transfer of bundles, request various additional and diagnostic information, an end-to-end acknowledgment, custody and several security features such authentication (signing), confidentiality (encryption) and error detection (signatures to detect modifications). The security-related options are optional and only apply if security is enabled. A listing of all available options can be found in Table 5.1. In response to bundles with some of the above mentioned options set, bundle status reports are generated. These provide information and diagnostic responses, corresponding to the ICMP protocol in IP [27]. However, in contrast to ICMP, bundles contain an additional field for a report-to EID in addition to source and destination EIDs. This report-to identifier may be different from the source identifier.

5.0.8 Custody Transfers

The most basic service provided by the bundle layer is unacknowledged unicast message delivery. The delivery reliability can be enhanced by requesting cus-

tody transfers. Custody transfer means moving the responsibility for reliable delivery of an ADU's bundles among different DTN nodes. This is similar to moving responsibility for email messages between different email servers using the SMTP protocol. A node accepting a custody transfer is called a *custodian*. It has to make sure that the bundles are stored in persistent storage and can only remove them once the custody has been successfully transferred to another node. If a node accepts a custody transfer, a Custody Transfer Accepted Signal is sent back to the previous custodian. The new custodian then updates the Custodian EID field in the respective bundle(s) before it is forwarded further. Note that not all nodes are required to accept a custody transfer. This may happen if e.g. a node would not have sufficient storage space. The decision of accepting a custody transfer is based on solving a resource allocation and scheduling problem. In general, applications do not have to request custody transfers. The successful delivery of bundles relies on the reliability mechanisms of the underlying protocols below the bundle layer. With custody transfer requested, the bundle layer provides an addition timeout and retransmission mechanism and a custodian-to-custodian bundle-layer acknowledgment scheme.

In a network with strictly one-directional custodian-to-custodian hops, the custody transfers will not be acknowledged as there is no way to back-signal the custody transfer acknowledgments. For this case, a mechanism is provided to ameliorate the incorrect information that a bundle has been lost. If the option "Report When Bundle Forwarded" is set, the nodes would report the existence of a known one-way path using a bundle status report.

5.0.9 Contact Types

The DTN architecture provides also a framework for routing and forwarding for unicast, anycast and multicast bundles. Links between nodes can have varying delay and capacity over time. Furthermore, some links be one-directional only. The period of time when a link's capacity is strictly positive is called a *contact*. If contacts and their capacities are known ahead of time, smart routing and forwarding decisions can be made. Handling situations with lossy delivery paths or unknown contact intervals or capacities are still an active research area. Based on predictability of performance characteristics, contacts can be divided into following categories:

persistent – always available, such as DSL or cable modem connections

on-demand – an action has to be taken to initiate contact, but then acts as a persistent contact. An example would be a dial-up connection.

intermittent - scheduled – the contact schedule is known ahead of time, such as with a low-earth orbiting satellite

intermittent - opportunistic – contacts presenting themselves unexpectedly, such as an aircraft flying by or a PDA passing by with bluetooth connection enabled. There is no pre-determined schedule for these contacts.

intermittent - predicted – based on on fixed schedule, but likely contact times can be predicted from history of contacts or other information.

5.0.10 Fragmentation

The DTN framework provides fragmentation and reassembly mechanisms to improve efficiency of bundle transfers by fully utilizing contact bandwidth and period and avoiding retransmission of partially transferred bundles. There are two forms of fragmentation, proactive and reactive. In *proactive fragmentation* a DTN node may divide an ADU into multiple bundles and transmit them independently. The final destination is then responsible for reassembling the complete ADU from the smaller bundles. This approach is used primarily when contacts are known in advance or can be predicted. With *reactive fragmentation* nodes may fragment a bundle cooperative when only part of it is transferred. The receiving node then modifies the bundle to indicate that it is a fragment and forwards it further as usual. The other node may learn that only a part of the bundle was transferred to the next hop and transmit the remaining portion of the bundle during subsequent contact opportunities. This may well happen via different next-hop nodes if routing changes. The reactive fragmentation is not required for every DTN implementation, but fragment reassembly is. Reactive fragmentation may pose significant challenges in case of digital signatures and authentication codes. In case DTN security is enabled, proactive fragmentation may have to be used.

Although of importance, the issues of congestion and flow control have not yet been resolved withing the DTNRG research group.

5.0.11 Time Synchronization

The DTN architecture depends on time synchronization between DTN nodes primary for the following reasons:

- bundle expiration time computations – Each bundle contains a creation timestamp and an explicit expiration field (number of seconds after creation) on each bundle. These are used to determine how long a bundle is valid and when it can be discarded.
- bundle and fragment identification – The concatenation of the creation timestamp and the source EID serves as a unique identifier for an ADU. Such identified is used by custody transfers and bundle fragments reassembly.
- routing with scheduled or predicted contacts
- application registration expiration – Application registrations for for receiving traffic for an EID are maintained only for a finite time, specified during the registration.

5.0.12 Security

The possibility of severe resource scarcity in some DTN networks requires some form of authentication and network access control. For example, it should not be possible for an unauthorized user to flood the network, possibly denying service to legitimate users. Furthermore, unauthorized traffic should not be forwarded

at all over some special, mission-critical links. For this purpose the DTN framework standardizes a security architecture . It utilizes both end-to-end and hop-by-hop authentication and integrity mechanisms. Using both approaches allows to handle access control for data forwarding separately from application-layer data integrity. While the end-to-end mechanisms may be used to authenticate principals such as users, the hop-by-hop mechanisms authenticate DTN nodes as legitimate bundle transceivers to each other. If authentication or access control checks fail, traffic is discarded as early as possible by the DTN nodes. The purpose for standardizing a DTN security architecture is that standard approaches have shortcomings due to the delays and disconnections in a DTN environment, making updating access control lists, revoking credentials or frequent accesses to an authentication server unattractive. Note that the security architecture is optional for DTN.

5.0.13 State Maintenance

Various types of state have to be managed by the bundle layer.

Application registration state is created by applications and removed by an explicit request or timeout. The state should be retained across application and system restarts. Due to the possibly high round-trip time, an application might have to be restarted when a response comes back. State information has to be maintained to enable a correct reinstantiation of the respective application.

A custodian has to keep account of bundles for which is has accepted custody. Additionally, protocol state related to transferring custody has to be maintained. Custody state information related to a bundle can be released when a Custody Transfer Succeeded signal is received, indicating that custody has been transferred to another node.

Information related to routing and forwarding has to be maintained. Bundles to be forwarded may stay in queues for considerable amounts of time while waiting for a communication opportunity. While unicast and anycast bundles may be discarded after a successful transfer to the next hop, multicast bundles constitute an additional burden as the have to be archived longer in case a registration for the multicast group arrived later.

In case the DTN security approach is enabled, additional state information needs to be maintained. This includes node's own private information, credentials and revocation lists, access control lists including updates and cached possibly public information and credentials of their next-hop neighbors.

Finally, each node has to maintain its own configuration and policy state.

As bundle delivery has to operate over networks with significant delays, applications using the DTN networks should be designed in a delay-tolerant fashion as well. Communication primitives provided by the DTN architecture are based on an asynchronous, message-based communication rather than a request-response model. ADUs created by an application should be sufficiently self-contained to be treated independently by the receiver rather than rely on information in other ADUs.

Due to the possibly long delays between sending a message and obtaining a response, an application may terminate before the response arrives. The application should be designed in a way allowing for easy reinstantiation using save state information from persistent storage.

5.0.14 Convergence Layer

As the DTN architecture uses for the underlying communication various different protocols offering varying functionality, additional per-protocol adaptation may be accomplished by a convergence layer between the bundle layer and the underlying protocol layer. The complexity of these convergence layers may differ across protocols, but would provide a consistent interface for the bundle layer. For example, for some protocols, the convergence layer would have to implement an acknowledgment scheme while other protocols, such as TCP/IP might already include it. The convergence layer for TCP/IP is defined in [10].

5.1 LTP

Another underlying protocol that can be used is the Licklider Protocol (LTP), which will be described in more detail. An overview of LTP can be found in [6]. LTP is intended as a reliable convergence layer over single-hop deep-space RF links, i.e. links with extremely long round trip times and/or frequent interruptions in connectivity, but can be applied in other environments as well. The basis of LTP design comes from the Consultative Committee for Space Data Systems (CCSD) File Delivery Protocol (CFDP). CFDP provides reliable file transfer across interplanetary distances by detecting loss and automatically retransmitting. CFDP itself, however, has only rudimentary built-in networking capabilities. LTP's design notions are directly descended from CFDP's retransmission procedures.

LTP is basically a point-to-point protocol between two antennae. Hence, it is assumed that the operating environment is able to pass information on the link status, the so called "link state cues" to LTP. This assumption is motivated by the interplanetary communication, where effort is spent on having the right antenna orientation and transmission power. Hence, LTP is informed when data should be transmitted and received. This allows for deferring transmission if there is no link. Furthermore, timers can be suspended during interrupted connectivity. The round trip times are assumed to be deterministic and are estimated from the distance between the two communication endpoints assuming signals not moving faster than at the speed of light.

Although LTP is a stateful protocol, it does not use any negotiation or handshakes before exchanging data. Typically long round-trip times result in having a rather large number of transmissions concurrently in flight. As the loss of transmission state due to rebooting or power cycling an LTP engine would result in rather costly retransmissions, transmission information is retained in non-volatile memory.

A single LTP association between two nodes can accommodate several concurrent sessions, one for each block of data in transit. As there are no multiple paths, it is assumed that packets cannot be reordered on the link. However, loss or corruption of packets can occur.

LTP provides partial reliability for data transmission. The application can mark which data is "red" and which is "green". Delivery of "red" data is then guaranteed by using acknowledgments while for the "green" data best effort delivery is used. The motivation is that some data is worthless without the corresponding header, but missing only part of the data is still OK. Technically,

each block of data contains a “red” and “green” part, where each can of zero length.

LTP sports laconic acknowledgments, where acknowledgments are aggregated into reception reports. These reports are sent only upon encountering specific solicitations for reception reports, so called “checkpoints”. The reception reports are mandatory at the end of “red” data and at the end of transmission. The operation of LTP is then to send segments, receive a report and acknowledge the reception of the report. Using the selective acknowledgments, LTP provides reliable communication.

Chapter 6

Proposed work

The aim of the proposed Master Thesis is to investigate connecting wireless sensor networks to the Internet. The hardware platform will be the MicaZ and TelosB motes from CrossBow Technologies running the TinyOS system. These motes support the 802.15.4 wireless communication standard, which will be used. IP connectivity and higher layer protocols, such as UCP, TCP and ICMP will have to be enabled on the motes. For this, the 6lowpan standard is envisioned to be used.

The focus of the work will be on investigating and evaluating mesh networking over 802.15.4 links on the motes. For this purpose, a test bed will be set up. While the 802.15.5 standard is aimed at 802.15.4 links, wireless sensor networks sport rather intermittent connectivity due to the nodes spending a significant portion of their life time in sleep states. For this purpose, other mesh networking protocols developed for MANETs might be investigated and insights from DTN can be applicable as well.

Bibliography

[1] Hari Balakrishnan, Srinivasan Seshan, Elan Amir, and Randy H. Katz. Improving TCI/IP performance over wireless networks. In *MobiCom '95: Proceedings of the 1st annual international conference on Mobile computing and networking*, pages 2–11, New York, NY, USA, 1995. ACM Press.

[2] Andreas Birk. Fast Robot Prototyping with the CubeSystem. In *Proceedings of the International Conference on Robotics and Automation, ICRA'2004*. IEEE Press, 2004.

[3] R. Braden. Requirements for Internet Hosts - Communication Layers. RFC 1122, IETF, October 1989.

[4] T. Braun, T. Voigt, and A. Dunkels. Energy-Efficient TCP Operation in Wireless Sensor Networks. *Praxis der Informationsverarbeitung und Kommunikation (PIK)*, 2005.

[5] S. Burleigh, A. Hooke, L. Torgerson, K. Fall, V. Cerf, B. Durst, K. Scott, and H. Weiss. Delay-tolerant networking: an approach to interplanetary Internet. *IEEE Communications Magazine*, 41(6):128–136, 2003.

[6] Scott C. Burleigh, M. Ramadas, and Stephen Farrell. Licklider Transmission Protocol - Motivation. Internet-Draft Version 03, IETF, September 2006.

[7] David D. Clark. Window And Acknowledgement Strategy in Tcp. RFC 813, IETF, July 1982.

[8] Matt Crawford. Transmission of IPv6 Packets over Ethernet Networks. RFC 2464, IETF, December 1998.

[9] Crossbow Technology, Inc. Mica2 Datasheet. http://www.xbow.com/Products/Product_pdf_files/Wireless_pdf/MICA2_Datasheet.pdf.

[10] M. Demmer. Delay Tolerant Networking TCP Convergence Layer Protocol. Internet-Draft Version 00, IETF, October 2006.

[11] A. Dunkels. Full TCP/IP for 8-bit architectures. In *In Proceedings of The First International Conference on Mobile Systems, Applications, and Services (MOBISYS '03)*, May 2003.

[12] A. Dunkels, J. Alonso, and T. Voigt. Making TCP/IP Viable for Wireless Sensor Networks. In *1st European Workshop on Wireless Sensor Networks (EWSN 2004)*, 2004.

[13] Adam Dunkels. Minimal TCP/IP implementation with proxy support. Master's thesis, Swedish Institute of Computer Science, February 2001.

[14] Stephen Farrell, Vinny Cahill, Dermot Geraghty, Ivor Humphreys, and Paul McDonald. When TCP Breaks: Delay- and Disruption- Tolerant Networking. *IEEE Internet Computing*, 10(4):72–78, 2006.

[15] D. Gay, P. Levis, R. von Behren, M. Welsh, E. Brewer, and D. Culler. The nesC Language: A Holistic Approach to Networked Embedded Systems. In *PLDI03*. ACM, June 2003.

[16] J. Hill, P. Bounadonna, and D. Culler. Active Message Communication for Tiny Network Sensorss. http://webs.cs.berkeley.edu/tos/papers/ammote.pdf.

[17] Robert M. Hinden and Stephen E. Deering. Internet Protocol Version 6 (IPv6) Addressing Architecture. RFC 3513, IETF, April 2003.

[18] IEEE. Guidelines for 64-Bit Global Identifier (Eui-64) Registration Authority.

[19] IEEE. IEEE standard for information technology - telecommunications and information exchange between systems - local and metropolitan area networks specific requirements part 15.4: wireless medium access control (MAC) and physical layer (PHY) specifications for low-rate wireless personal area networks (LR-WPANs), 2003.

[20] IEEE. IEEE P802.15.5 Draft Candidate, November 2005.

[21] IEEE. Preliminary Draft of Baseline Document for 802.15.5 Mesh Networking, July 2006.

[22] Interplanetary Internet. http://www.ipnsig.org/.

[23] Nandakishore Kushalnagar and Gabriel Montenegro. 6LoWPAN: Overview, Assumptions, Problem Statement and Goals. Internet-Draft Version 05, IETF, August 2006.

[24] X. Luo, K. Zheng, Y. Pan, and Z. Wu. A TCP/IP implementation for wireless sensor networks. In *IEEE International Conference on Systems, Man, and Cybernetics*, 2004.

[25] Gabriel Montenegro and Nandakishore Kushalnagar. Transmission of IPv6 Packets over IEEE 802.15.4 Networks. Internet-Draft Version 06, IETF, November 2006.

[26] Thomas Narten, Erik Nordmark, William Allen Simpson, and Hesham Soliman. Neighbor Discovery for IP version 6 (IPv6). Internet-Draft Version 09, IETF, October 2006.

[27] J. Postel. Internet Control Message Protocol. RFC 792, IETF, September 1981.

[28] Behcet Sarikaya. Serial forwarding approach to connecting TinyOS-based sensors to IPv6 Internet. Internet-Draft Version 00, IETF, February 2006.

[29] Sendt. http://down.dsg.cs.tcd.ie/sendt/. Web Page.

[30] Kannan Srinivasan, Prabal Dutta, Arsalan Tavakoli, and Philip Levis. Some Implications of Low Power Wireless to IP Networking. In *Fifth Workshop on Hot Topics in Networks (HotNets-V)*, 2006.

[31] Vinton G. Cerf and Scott C. Burleigh and Robert C. Durst and Dr. Kevin Fall. Delay-Tolerant Network Architecture. Internet-Draft Version 07, IETF, October 2006.

[32] ZigBee Alliance. http://www.zigbee.org/.